Claims (clean version)

44. A processor device comprising

an instruction stream transformation unit that transforms code blocks of instructions from an original instruction set architecture to a transformed instruction set architecture,

a regular cache that stores instructions in said original instruction set architecture,

an instruction stream cache that stores instructions in said transformed instruction set architecture, and

an execute unit for executing instructions,

wherein said instruction stream transformation unit transforms said code blocks of instructions from said original instruction set architecture to said transformed instruction set architecture,

wherein said instruction stream cache stores said code blocks after transformation to said transformed instruction set architecture for possible future execution,

wherein said instruction stream cache is addressed by some of said fetch requests from said execute unit and can potentially respond to some of said fetch requests for said code blocks after transformation without requiring cache hit information from said regular cache after said code blocks have already been transformed and stored into the instruction stream cache, and

whereby the execution of a program code by said processor device is accelerated by transforming portions of said program code at run-time into the transformed instruction set architecture for more efficient execution and caching the transformed code within the instruction stream cache for possible repeated execution without requiring repeated transformations.

45. A processor device as in claim 44 wherein

said execute unit can directly execute instructions in both said original instruction set architecture and said transformed instruction set architecture,

whereby the execute unit can execute code in said original instruction set architecture without having to first wait for said code to be transformed into said transformed instruction set architecture.

46. A processor device as in claim 45 wherein said execute unit is an in-order execute unit that retires instructions and commits their results in the same order as the instructions occur in the code.

47. A processor device as in claim 45 wherein said execute unit is a dynamic out-of-order execute unit that can retire instructions out of order compared to the order of the instructions in the code.

48. A processor device as in claim 44 further comprising a working memory connected to said instruction stream transformation unit for storing intermediate calculations during the process of transforming code from said original instruction set architecture into said transformed instruction set architecture.

49. A processor device as in claim 44 wherein said instruction stream transformation unit transforms blocks of code which are presumed hyper-blocks.

50. A processor device as in claim 44 wherein said instruction stream transformation unit transforms blocks of code which are denoted as hyper-blocks in original instruction set architecture code.

51. A processor device as in claim 44 wherein said instruction stream cache comprises means of using a tag for each cache line to denote whether the tag is a start of a hyper-block.

52. A processor device as in claim 44 wherein said instruction stream cache comprises means of using a hyper-block ID as an alternative way of addressing a transformed block of code.

53. A processor device as in claim 52 wherein said instruction stream cache comprises means of storing a plurality of hyper-block lines that are chained together using a common hyper-block ID plus pointers.

54. A processor device as in claim 44 wherein said instruction stream cache only hits when a transformed block of code is entered starting from the first instruction of the block.

55. A processor device as in claim 44 wherein said instruction stream transformation unit comprises means of instruction re-ordering.

56. A processor device as in claim 44 wherein said instruction stream transformation unit comprises means of performing predication if-conversion to convert an if-then-else construct in said code into a predication calculation and a series of predicated instructions that conditionally commit their results depending on the results of said predication calculation.

57. A processor device as in claim 44 wherein said instruction stream transformation unit comprises means of converting a load instruction into a speculative load and a load activation instruction pair,

whereby the possibility of more efficient scheduling of the transformed code is enabled by allowing the speculative load to be scheduled earlier than a normal load could be scheduled without this conversion thus minimizing possible waiting during execution for this memory load to complete.

58. A processor device as in claim 44 wherein said instruction stream transformation unit comprises a parallel dependency detector circuit for detecting possible dependencies between instructions,

whereby said instruction stream transformation unit can efficiently detect said possible dependencies during the process of instruction stream transformation.

59. A processor device as in claim 44 wherein said instruction stream transformation unit transforms said code blocks by dividing long instruction sequences into sequences of a defined maximum number of instructions called an instruction window that is equal to a number of instructions that said instruction stream transformation unit can work with effectively.

60. A processor device as in claim 59 wherein said instruction stream transformation unit transforms said code blocks by dividing long instruction sequences into overlapping sequences of a defined maximum number of instructions called overlapping instruction windows,

whereby this enables the efficient scheduling of code both within the middle of instruction windows and within the overlap regions between instruction windows.

61. A processor device as in claim 44 wherein said instruction stream transformation unit comprises means of creating a dependency matrix to represent potential dependencies between instructions,

whereby the dependency matrix provides dependency information in an efficiently accessed manner for the instruction stream transformation unit.

62. A processor device as in claim 61 wherein said instruction stream transformation unit comprises means of creating an operand mapping table to represent potential writes of operands by instructions,

whereby said operand mapping table enables dependencies between instructions to be detected efficiently and for the dependency matrix to be built efficiently by said instruction stream transformation unit.

63. A processor device as in claim 62 wherein said instruction stream transformation unit comprises means of performing register renaming to rename uses of registers in the transformed code blocks when necessary to minimize write-after-write hazards between write instructions,

whereby scheduling dependencies caused by said write-after-write hazards is reduced.

64. A processor device as in claim 63 wherein said register renaming by the instruction stream transformation unit allocates a set of physical registers that are separate from a second set of registers allocated by dynamic register renaming done by the execute unit,

whereby said instruction stream transformation unit and said execute unit are each able to independently allocate physical registers for register renaming without conflicting with each other.

65. A processor device as in claim 44 wherein the instruction stream transformation unit comprises means of performing an instruction scheduling algorithm to schedule a code block.

66. A processor device as in claim 65 wherein said scheduling algorithm is a list scheduling algorithm comprising the steps of

doing a basic forward iterative traversal to calculate the minimum cycle number of each instruction in said block from the start of said block,

propagating the depth of each leaf instruction to all its predecessors to calculate each instruction's priority as defined by the depth of its deepest child, and

performing a second forward iterative traversal to schedule instructions for execution cycle-by-cycle, where the scheduling priorities are used in conjunction with a ready list of instructions that are ready to be scheduled because all dependencies have been resolved.

67. A processor device as in claim 44 wherein

said execute unit comprises means of performing dynamic memory disambiguation,

said transformed instruction set architecture supports notations for indicating ambiguous memory operations,

and said instruction stream transformation unit calculates said notations for indicating ambiguous memory operations.

68. A processor device as in claim 67 wherein said instruction stream transformation unit comprises means of creating an ambiguous memory dependency matrix.

69. A processor device as in claim 67 wherein said instruction stream transformation unit comprises means of converting an ambiguous memory read instruction into a speculative read and a read check instruction pair.

70. A processor device as in claim 44 comprising a dynamically-scheduled execute unit.

71. A processor device as in claim 44

wherein the transformed instruction set architecture comprises dependency notation means of explicitly describing dependencies between instructions, and

wherein the instruction stream transformation unit calculates dependency notations for said code blocks during the process of transforming said code blocks into said transformed instruction set architecture,

whereby the transformed code blocks can be executed without having to redetect all static dependencies.

72. A processor device as in claim 71 wherein said means of explicitly describing dependency notation means allows instructions to be grouped into mini-tuples for dependency notations.

73. A processor device as in claim 71 wherein said dependencies are represented by dependency pointers in the transformed instruction set architecture.

74. A processor device as in claim 71 wherein said dependencies are represented by dependency vectors in the transformed instruction set architecture.

75. A processor device as in claim 44 further comprising means to perform semi-dynamic instruction code re-writing and re-scheduling,

whereby the code can be further optimized based on run-time information compared to code that is transformed only once through the instruction stream transformation unit.

76. A processor device as in claim 44 further comprising at least one run-time history table.

77. A processor device as in claim 76 comprising a value prediction history table to record the history of success or failure of previous executions of value predictions,

whereby run-time behavior about value predictions can be recorded in order to help optimize subsequent instruction scheduling.

78. **(amended)** A processor device

that executes an instruction set architecture comprising

a predicate calculation instruction and

a predicated instruction that conditionally executes depending on the results of a previous predicate calculation wherein said predicated instruction is not a conditional branch instruction,

wherein said processor device comprises a predicate history table to record the history of previous executions of predicate calculation instructions,

whereby run-time behavior about predicates can be recorded in order to help optimize subsequent instruction scheduling.

79. **(amended)** A processor device comprising

a data cache,

a data hit/miss history table to record the history of whether previous executions of memory access instructions were hits or misses in said data cache,

and an execute unit for executing instructions that uses information from said data hit/miss history table as part of a process of scheduling said instructions prior to beginning execution of said instructions,

whereby run-time behavior about data hit/misses can be recorded in order to help optimize subsequent instruction scheduling.

80. **(amended)** A processor device as in claim 76 comprising an ambiguous memory conflict history table to record the history of whether previous executions of promoted ambiguous read instructions cause memory conflicts or not,

whereby run-time behavior about ambiguous memory conflicts can be recorded in order to help optimize subsequent instruction scheduling.

81. **(amended)** A method of providing precise interrupts in a processor implementing instruction stream transformation comprising the steps of

mapping an original instruction set architecture instruction to an equivalent group of one or more transformed instruction set architecture instruction(s) that include explicit notations to indicate static register renaming from logical registers to physical registers,

using said physical registers that have not been committed to said logical registers to hold results, and

allowing the final instruction in said group to commit the physical register result(s) to logical register(s).

82. **(amended)** A method of providing precise interrupts in a processor implementing instruction stream transformation comprising the steps of

assigning an instruction sequence number to each original instruction set architecture instruction starting from the beginning of the code block,

using an explicit field within each transformed instruction set architecture instruction to record the instruction sequence number of the corresponding original instruction set architecture instruction, and

committing the results of instructions in order of the instruction sequence numbers.

83. **(amended)** A software method of performing code scheduling comprising the steps of

reading a sequence of instructions from memory,

building a dependency matrix to represent potential dependencies between said instructions,

computing a scheduling of said instructions, and

writing a re-ordered sequence of said instructions to memory for later execution,

whereby the dependency matrix provides dependency information in an efficiently accessed manner for said software method to perform the code scheduling.

84. **(amended)** A processor device comprising

means of executing instructions from an instruction set architecture

wherein the format of the instruction set architecture comprises fields containing dependency vector(s) to explicitly note dependencies between instructions.

85. **(amended)** A processor device comprising

means of executing instructions from an instruction set architecture

wherein the format of the instruction set architecture comprises fields containing dependency pointer(s) to explicitly note dependencies between instructions.

86. Cancelled.